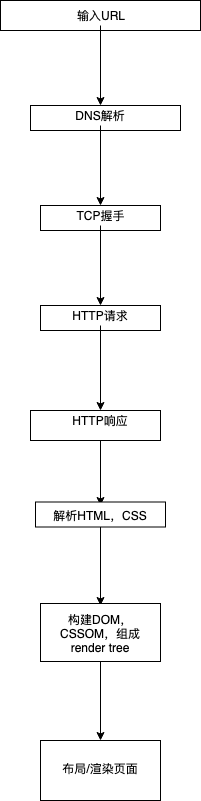
**一次完整的web请求和渲染过程以及如何优化网页**

本文主要介绍访问一个网站时的流程，页面渲染过程，包括其中涉及到的概念。 以及，我们如何去优化前端页面，让它访问速度更快。（critical rendering path最佳渲染路径）

1. 打开浏览器，输入URL
2. DNS解析
3. 完成TCP握手
4. 发送HTTP请求
5. 接收HTTP响应结果
6. 浏览器解析HTML，CSS
7. 构建对象模型，DOM tree 和 CSSOM tree，组成render tree
8. 浏览器渲染页面(布局)



## HTTP1.x和HTTP2

在 HTTP/1.x 中，如果客户端要想发起多个并行请求以提升性能，则必须使用多个 TCP 连接。 这是 HTTP/1.x 交付模型的直接结果，该模型可以保证每个连接每次只交付一个响应（响应排队）。 更糟糕的是，这种模型也会导致队首阻塞，从而造成底层 TCP 连接的效率低下。 也就是说在目前的HTTP1.X的协议下，浏览器对资源的并发请求个数是有限制的。 等到HTTP2到来的时候，通过二进制分帧层进行优化。 HTTP/2 中新的二进制分帧层突破了这些限制，实现了完整的请求和响应复用：客户端和服务器可以将 HTTP 消息分解为互不依赖的帧，然后交错发送，最后再在另一端把它们重新组装起来。

优势：

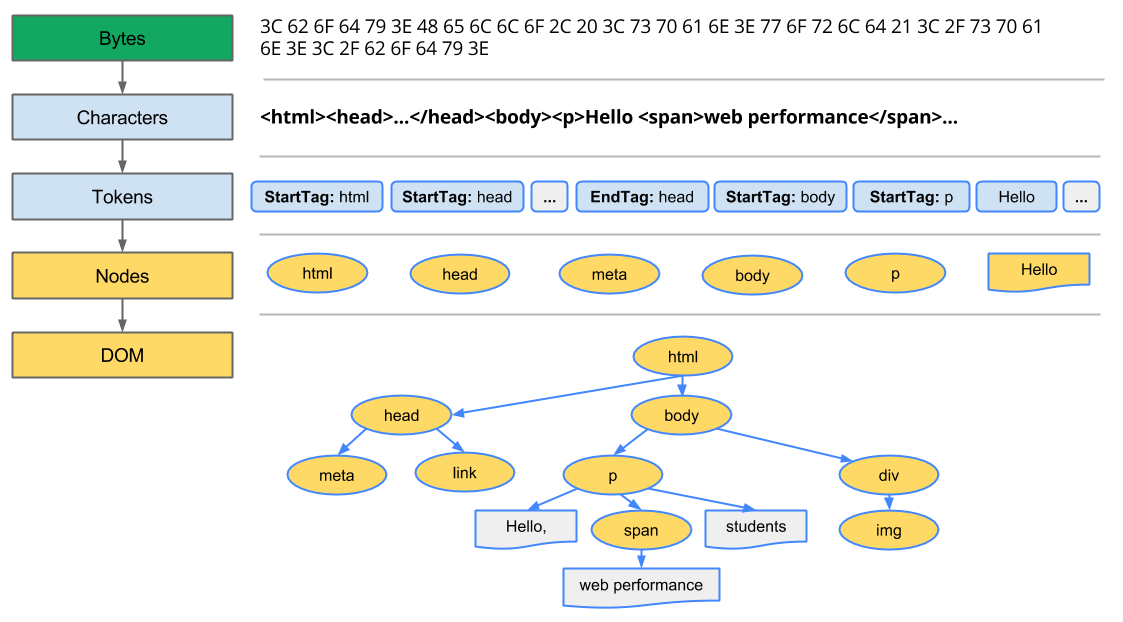
* 并行交错地发送多个请求，请求之间互不影响。
* 并行交错地发送多个响应，响应之间互不干扰。
* 使用一个连接并行发送多个请求和响应。

但这些并不是我们能够优化的部分。

基于现在的网络条件，我们可以采用多个CDN地址，进行不同源的并发改善优化。 这篇文章主要介绍，第六步开始我们可以优化的部分，也就是解析HTML，CSS。

## 浏览器是如何构建对象模型的？

* 字节 → 字符 → 令牌 → 节点 → 对象模型。
* HTML标记与CSS标记都会经历上述过程，HTML变为DOM，CSS变为CSSOM



## render tree构建，布局以及绘制

构建好render tree之后，会过滤掉display:none 这种无需渲染的节点。将tree渲染到页面。

值得注意的是，每一次的dom或者造成布局影响的变动，都会触发 reflow(回流/重排)，会消耗很大的页面资源。

reflow是由于dom或者布局的变动而触发，如修改了dom位置，或是宽高，margin, padding等。 repaint是样式风格修改，不影响布局时触发，如改了颜色之类的。

提高网页渲染速度，主要可以减少 DOM, CSSOM处理， 合并render tree，以及 减少reflow的次数。

## 对CSS的优化

根据上面的内容我们知道了，想要渲染页面必须有render tree，而render tree是由DOM tree以及CSSOM tree组成的。

* 默认下，CSS会阻塞渲染页面
* 我们通过@media等，可以让CSS标记为不阻塞渲染
* 不论是否阻塞渲染，浏览器都会将CSS资源下载到客户端

所以，为了让页面更快的渲染，\*我们必须要尽早的将CSS资源下载到我们的客户端。\*以及使用@media进行优化

为了获得最佳性能，你可以使用一些inline css，这样不会去CDN获取资源，从而造成多次往返

## 对JS的优化

当 HTML 解析器遇到一个 script 标记时，它会暂停构建 DOM，将控制权移交给 JavaScript 引擎；等 JavaScript 引擎运行完毕，浏览器会从中断的地方恢复 DOM 构建。

这也就是为什么我们需要将script tag放在页面的底部。 /React APP一般会如此使用。因为React需要将整个APP 渲染到一个DOM节点上，如果放置在DOM之上，会造成React找不到该渲染的节点，从而报错/

而我们一般不建议在render tree刚刚建立的时候，就使用JS去操作DOM，从而造成reflow，也就是说，希望JS不要成为render tree的一部分。这个时候，我们将<script> 放在页面靠下的部分就可以不阻塞页面的渲染。

如果你的<script>是从CDN获取资源，那么等待的过程也会造成一定的阻塞。

你可以将<script>加上 async，让它变为异步加载。

为了实现最佳性能，建议去除关键渲染路径中任何不必要的JavaScript

/关键渲染路径是指优先显示与当前用户操作有关的内容。/